Data Aggregation and Group Operations

CS 3753 Data Science

Prof. Weining Zhang

Topics

- Grouping rows: various ways to specify grouping criteria
- Grouping columns
- Data Aggregation: applying functions to each group
- Combine aggregation into Series or DataFrames
- Pivot tables and cross tabulation
- Examples of data transformation, data reduction, and dada analysis

In [ ]:
```
from __future__ import division
from numpy.random import randn
import numpy as np
import os
import matplotlib.pyplot as plt
np.random.seed(12345)
plt.rc('figure', figsize=(10, 6))
from pandas import Series, DataFrame
import pandas as pd
np.set_printoptions(precision=4)

pd.options.display.notebook_repr_html = False

%matplotlib inline
```

Pandas GroupBy Mechanics

The Pandas GroupBy (https://pandas.pydata.org/pandas-docs/stable/groupby.html) referring to a process involving one or more of the following steps

- Splitting rows into groups based on some criteria
- Applying a function to each group independently
- Combining the results into a data structure

Types of Functions

- Aggregation: computing a summary statistic (or statistics) about each group. Some examples:
  - Compute group sums or means
  - Compute group sizes / counts
- Transformation: perform some group-specific computations and return a like-indexed. Some examples:
  - Standardizing data (z-score) within group
  - Filling NAs within groups with a value derived from each group
- Filtering: discard some groups, according to a group-wise computation that evaluates True or False. Some examples:
  - Discarding data that belongs to groups with only a few members
  - Filtering out data based on the group sum or mean
- Some combination of the above: GroupBy will examine the results of the apply step and try to return a sensibly combined result if it doesn't fit into either of the above two categories

Create Groups

```
df[<value-columns>].groupby(by=None, axis=0,...)
```

- by can be a mapping, function, label, or list of labels
  - Series, dict, nparray can be used to identify rows for groups
  - function can be applied to index to determine groups
  - labels specify columns used to determine groups
- axis=0: group rows into groups, axis=1: group columns into groups

In [ ]:
```
df = DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                'key2' : ['one', 'two', 'one', 'two', 'one'],
                'data1' : np.random.randn(5),
                'data2' : np.random.randn(5)})
df
```

In [ ]:
```
grouped = df['data1'].groupby(df['key1'])
grouped.groups
# Use grouped.<TAB> to show attributes and fuctions for GroupBy object
```

In [ ]:
```
df.groupby(df['key1']).get_group('b')
```

Apply Simple Function to Groups

```
df.groupby(...).<function>()
```

- predefined functions include

```
mean(), min(), max(), count()
```

- The result can be a Series or DataFrame

In [ ]:
```
grouped.mean()
```

In [ ]:
```
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
```

In [ ]:
```
means.unstack()
```

In [ ]:
```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
years = np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([states, years]).mean()
df.groupby([states, years]).mean()
```

In [ ]:
```
df.groupby('key1').mean()
df.groupby(['key1', 'key2']).mean()
df.groupby(['key1', 'key2']).groups
```

In [ ]:
```
df.groupby(['key1', 'key2']).size().unstack()
```

Iterating Over Groups

Explicit for-loop over groups

In [ ]:
```
print(df.groupby('key1').groups)
```

In [ ]:
```
for name, group in df.groupby('key1'):
    print(name)
    print(group)
```

In [ ]:
```
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
```

In [ ]:
```
dict([('a', 1),('b', 2)])
```

In [ ]:
```
pieces = dict(list(df.groupby('key1')))
pieces['b']
pieces
```

In [ ]:
```
df.dtypes
```

In [ ]:
```
grouped = df.groupby(df.dtypes, axis=1)
dict(list(grouped))
#dict(list(grouped))[df.dtypes['key1']]
```

Selecting a Subset of Columns

Either selecting columns before grouping or selecting columns after grouping. Some options are

```
df.groupby('key1')['data1']
df.groupby('key1')[['data2']]
df['data1'].groupby(df['key1'])
df[['data2']].groupby(df['key1'])
```

In [ ]:
```
df.groupby(['key1', 'key2'])[['data2']].mean()
```

Creating Column Groups

- For each row, divide columns into groups
- Explicitly identify groups by using Dict or Series
- Specify axis=1

Example

For each row, separate columns into groups red and blue, calculate the sum of the red columns and sum of blue columns, using Dict or Series to group columns

In [ ]:
```
people = DataFrame(np.random.randn(5, 5),
                   columns=['a', 'b', 'c', 'd', 'e'],
                   index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
people.loc[2:3, ['b', 'c']] = np.nan # Add a few NA values
people
```

In [ ]:
```
# Use dict to assign colors to column labels
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
           'd': 'blue', 'e': 'red', 'f' : 'orange'}
```

In [ ]:
```
# Grouping columns
by_column = people.groupby(mapping, axis=1)
by_column.sum()
```

In [ ]:
```
# Use Series to assign colors to columns
map_series = Series(mapping)
map_series
```

In [ ]:
```
people.groupby(map_series, axis=1).count()
```

Grouping by Functions

Create groups based on the result values from applying a function index

In [ ]:
```
people
```

In [ ]:
```
# group by length of the index
people.groupby(len).sum()
```

In [ ]:
```
# group index first by length and then by key_list
key_list = ['one', 'one', 'one', 'two', 'two']
people.groupby([len, key_list]).min()
people.groupby([len, key_list]).groups
```

Grouping by Index Levels

In [ ]:
```
# Multi-level column labels
columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'],
                                     [1, 3, 5, 1, 3]], names=['city', 'tenor'])
hier_df = DataFrame(np.random.randn(4, 5), columns=columns)
hier_df
```

In [ ]:
```
hier_df.groupby(level='city', axis=1).count()
```

Data Aggregation

- Use agg() or aggregate() function can apply one or more functions on each group

In [ ]:
```
df
```

In [ ]:
```
# For each group in 'key1', find 90 percentile m+ (M-m)x0.9
grouped = df.groupby('key1')
grouped['data1'].quantile(0.9)
```

In [ ]:
```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

In [ ]:
```
grouped.describe()
```

Apply Multiple Functions

- Multiple aggregate function can be applied to one or more columns in each group
  - several functions can be applied on one column

In [ ]:
```
tips = pd.read_csv('../week07/ch08/tips.csv')
tips
```

Exercise

Find the average percentage of the tips based on 'sex' and 'smoker'

In [ ]:
```
# Add tip percentage of total bill
tips['tip_pct'] = tips['tip'] / tips['total_bill']
tips[:6]
```

In [ ]:
```
grouped = tips.groupby(['sex', 'smoker'])
grouped.groups
```

In [ ]:
```
# one function on one column
grouped_pct = grouped['tip_pct']
grouped_pct.agg('mean').unstack()
#grouped_pct.agg('mean')
```

Exercise

Find the mean, standard deviation, and the difference between max and min percentage of tips for each sex and smoker group

In [ ]:
```
# three functions on one column
grouped_pct.agg(['mean', 'std', peak_to_peak])
```

In [ ]:
```
# specify the output column name and function
grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

Exercise

For each sex-smoker group, find the count, mean and max of tip percentage and for total bill amount

In [ ]:
```
# apply multiple functions to each column
functions = ['count', 'mean', 'max']
result = grouped['tip_pct', 'total_bill'].agg(functions)
result
```

In [ ]:
```
result['tip_pct']
```

In [ ]:
```
# Use tuples to specify multiple functions
ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]
grouped['tip_pct', 'total_bill'].agg(ftuples)
```

In [ ]:
```
# use dict to specify multiple unctions
grouped.agg({'tip' : np.max, 'size' : 'sum'})
```

In [ ]:
```
grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
             'size' : 'sum'})
```

Returning Aggregated Data in "Unindexed" Form

In [ ]:
```
tips.groupby(['sex', 'smoker'], as_index=False).mean()
```

Group-wise Operations and Transformations

Example

Add two new columns to DataFrame df, one column contains the group mean for values in column 'data1', and the other column contains the group mean for values in column 'data2'

In [ ]:
```
df
```

In [ ]:
```
k1_means = df.groupby('key1').mean().add_prefix('mean_')
k1_means
```

In [ ]:
```
pd.merge(df, k1_means, left_on='key1', right_index=True)
```

Group-Specific Data Transformation

The transform() function will apply a function on each value in the DataFrame. It can be combined with groupby() function to apply group-specific transformations to values.

Example

- Group rows in people table by a list key, calculate group means for each column.
- Replace each value by its group mean. Subtract the group mean from each value

In [ ]:
```
people
```

In [ ]:
```
key = ['one', 'two', 'one', 'two', 'one']
people.groupby(key).mean()
```

In [ ]:
```
people.groupby(key).transform(np.mean)
```

In [ ]:
```
def demean(arr):
    return arr - arr.mean()
demeaned = people.groupby(key).transform(demean)
demeaned
```

In [ ]:
```
demeaned.groupby(key).mean()
```

General Split-Apply-Combine

- Use apply() function
  - `df.apply(func, axis=...)`
  - `groups.apply(func, axis=...)`
- axis=0: apply function to each column
- axis=1: apply funciton to each row

In [ ]:
```
# show top n (default to 5) values
def top(df, n=5, column='tip_pct'):
    return df.sort_values(by=column)[-n:]
top(tips, n=6)
```

In [ ]:
```
tips.groupby('smoker').apply(top)
```

In [ ]:
```
tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
```

In [ ]:
```
result = tips.groupby('smoker')['tip_pct'].describe()
result
```

In [ ]:
```
result.unstack('smoker')
```

In [ ]:
```
grouped.groups
```

In [ ]:
```
f = lambda x: x.describe()
grouped.apply(f)
```

Suppressing the Group Keys

In [ ]:
```
tips.groupby('smoker', group_keys=False).apply(top)
```

Example: Quantile and Bucket Analysis

- Fill a DataFrame of 1000 rows and 2 columns with random values.
- Create 4 equi-width bins on column 'data1'and find min, max, count and mean on 'data2' for each bin
- Create 10 equi-depth bins on column 'data1' and find min, max, count and mean on 'data2' for each bin

In [ ]:
```
frame = DataFrame({'data1': np.random.randn(1000),
                   'data2': np.random.randn(1000)})
factor = pd.cut(frame.data1, 4)
factor[:10]
```

In [ ]:
```
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}

grouped = frame.data2.groupby(factor)
grouped.apply(get_stats).unstack()

#ADAPT the output is not sorted in the book
# while this is the case now (swap first two lines)
```

In [ ]:
```
# Return quantile numbers
#grouping = pd.qcut(frame.data1, 10, labels=False)
grouping = pd.qcut(frame.data1, 10)

grouped = frame.data2.groupby(grouping)
grouped.apply(get_stats).unstack()
```

Example: Filling Missing Values with Group-specific Values

- Create a Series with some null values.
- Create two goups of the values and fill the NaN values in each group with the mean of that group

In [ ]:
```
s = Series(np.random.randn(6))
s[::2] = np.nan
s
```

In [ ]:
```
s.fillna(s.mean())
```

In [ ]:
```
states = ['Ohio', 'New York', 'Vermont', 'Florida',
          'Oregon', 'Nevada', 'California', 'Idaho']
group_key = ['East'] * 4 + ['West'] * 4
data = Series(np.random.randn(8), index=states)
data[['Vermont', 'Nevada', 'Idaho']] = np.nan
data
```

In [ ]:
```
data.groupby(group_key).mean()
```

In [ ]:
```
fill_mean = lambda g: g.fillna(g.mean())
data.groupby(group_key).apply(fill_mean)
```

In [ ]:
```
fill_values = {'East': 0.5, 'West': -1}
fill_func = lambda g: g.fillna(fill_values[g.name])

data.groupby(group_key).apply(fill_func)
```

Example: Random Sampling and Permutation

- Randomly draw a number of cards from a deck of cards
- Randomly draw a number of cards from each suit of cards

In [ ]:
```
# Hearts, Spades, Clubs, Diamonds
suits = ['H', 'S', 'C', 'D']
card_val = (list(range(1, 11)) + [10] * 3) * 4
base_names = ['A'] + list(range(2, 11)) + ['J', 'K', 'Q']
cards = []
for suit in ['H', 'S', 'C', 'D']:
    cards.extend(str(num) + suit for num in base_names)

deck = Series(card_val, index=cards)
deck
```

In [ ]:
```
deck[:13]
```

In [ ]:
```
def draw(deck, n=5):
    return deck.take(np.random.permutation(len(deck))[:n])
draw(deck)
```

In [ ]:
```
get_suit = lambda card: card[-1] # last letter is suit
deck.groupby(get_suit).apply(draw, n=2)
```

In [ ]:
```
# alternatively
deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
```

Example: Group-wise Weighted Average

In [ ]:
```
df = DataFrame({'category': ['a', 'a', 'a', 'a', 'b', 'b', 'b', 'b'],
                'data': np.random.randn(8),
                'weights': np.random.rand(8)})
df
```

In [ ]:
```
grouped = df.groupby('category')
get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
grouped.apply(get_wavg)
```

Example: Grouped Correlation

Find correlations between Apple, Microsoft, Exsson Mobile stocks and the S&P 500 Index

In [ ]:
```
close_px = pd.read_csv('ch09/stock_px.csv', parse_dates=True, index_col=0)
close_px.info()
```

In [ ]:
```
close_px[-4:]
```

In [ ]:
```
rets = close_px.pct_change().dropna()
spx_corr = lambda x: x.corrwith(x['SPX'])
by_year = rets.groupby(lambda x: x.year)
by_year.apply(spx_corr)
```

In [ ]:
```
# Annual correlation of Apple with Microsoft
by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

Example: Group-wise Linear Regression

Apply linear regression analysis on each group

In [ ]:
```
import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
```

In [ ]:
```
by_year.apply(regress, 'AAPL', ['SPX'])
```

Pivot Tables

A pivot table is a table that summarizes data in another table.

```
- df.pivot_table(<columns>,
                 index=...,
                 columns=...,
                 aggfunc=...
                 margines=...)
```

- Default aggregation function is mean()

Example

Consider tips dataset and use pivot_table to show various summarized info

In [ ]:
```
tips
```

In [ ]:
```
tips.pivot_table(index=['sex', 'smoker'])
```

In [ ]:
```
tips.pivot_table(['tip_pct', 'size'], index=['sex', 'day'],
                 columns='smoker')
```

In [ ]:
```
# Add marginal row and column
tips.pivot_table(['tip_pct', 'size'], index=['sex', 'day'],
                 columns='smoker', margins=True)
```

In [ ]:
```
# Use len() as aggregate function
tips.pivot_table('tip_pct', index=['sex', 'smoker'], columns='day',
                 aggfunc=len, margins=True)
```

In [ ]:
```
tips.pivot_table('size', index=['time', 'sex', 'smoker'],
                 columns='day', aggfunc='sum', fill_value=0)
```

Cross-Tabulations: crosstab

A contingency table (also known as a cross tabulation or crosstab) displays the (multivariate) frequency distribution in a DataFrame

```
df.crosstab(index, columns, aggfunc, margins,...)
```

- Default aggfunc is count

In [ ]:
```
from io import StringIO
data = """\
Sample    Gender    Handedness
1    Female    Right-handed
2    Male    Left-handed
3    Female    Right-handed
4    Male    Right-handed
5    Male    Left-handed
6    Male    Right-handed
7    Female    Right-handed
8    Female    Left-handed
9    Male    Right-handed
10    Female    Right-handed"""
data = pd.read_table(StringIO(data), sep='\s+')
```

In [ ]:
```
data
```

In [ ]:
```
pd.crosstab(data.Gender, data.Handedness, margins=True)
```

In [ ]:
```
pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
```

Exercise

Write Python code to create a table that represents the joint probability distribution of the three random variables: time, day, and smoker.

Example: 2012 Federal Election Commission Database

In [ ]:
```
fec = pd.read_csv('ch09/P00000001-ALL.csv')
```

In [ ]:
```
fec.info()
```

In [ ]:
```
fec.ix[123456]
```

In [ ]:
```
unique_cands = fec.cand_nm.unique()
unique_cands
```

In [ ]:
```
unique_cands[2]
```

In [ ]:
```
parties = {'Bachmann, Michelle': 'Republican',
           'Cain, Herman': 'Republican',
           'Gingrich, Newt': 'Republican',
           'Huntsman, Jon': 'Republican',
           'Johnson, Gary Earl': 'Republican',
           'McCotter, Thaddeus G': 'Republican',
           'Obama, Barack': 'Democrat',
           'Paul, Ron': 'Republican',
           'Pawlenty, Timothy': 'Republican',
           'Perry, Rick': 'Republican',
           "Roemer, Charles E. 'Buddy' III": 'Republican',
           'Romney, Mitt': 'Republican',
           'Santorum, Rick': 'Republican'}
```

In [ ]:
```
fec.cand_nm[123456:123461]
```

In [ ]:
```
fec.cand_nm[123456:123461].map(parties)
```

In [ ]:
```
# Add it as a column
fec['party'] = fec.cand_nm.map(parties)
```

In [ ]:
```
fec['party'].value_counts()
```

In [ ]:
```
(fec.contb_receipt_amt > 0).value_counts()
```

In [ ]:
```
fec = fec[fec.contb_receipt_amt > 0]
```

In [ ]:
```
fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack', 'Romney, Mitt'])]
```

Donation Statistics by Occupation and Employer

In [ ]:
```
fec.contbr_occupation.value_counts()[:10]
```

In [ ]:
```
occ_mapping = {
   'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
   'INFORMATION REQUESTED' : 'NOT PROVIDED',
   'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',
   'C.E.O.': 'CEO'
}

# If no mapping provided, return x
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)
```

In [ ]:
```
emp_mapping = {
   'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
   'INFORMATION REQUESTED' : 'NOT PROVIDED',
   'SELF' : 'SELF-EMPLOYED',
   'SELF EMPLOYED' : 'SELF-EMPLOYED',
}

# If no mapping provided, return x
f = lambda x: emp_mapping.get(x, x)
fec.contbr_employer = fec.contbr_employer.map(f)
```

In [ ]:
```
by_occupation = fec.pivot_table('contb_receipt_amt',
                                index='contbr_occupation',
                                columns='party', aggfunc='sum')
by_occupation
```

In [ ]:
```
over_2mm = by_occupation[by_occupation.sum(1) > 2000000]
over_2mm
```

In [ ]:
```
over_2mm.plot(kind='barh')
```

In [ ]:
```
def get_top_amounts(group, key, n=5):
    totals = group.groupby(key)['contb_receipt_amt'].sum()

    # Order totals by key in descending order
    # return totals.order(ascending=False)[-n:]
    return totals.sort_values(ascending=False)[-n:]
```

In [ ]:
```
grouped = fec_mrbo.groupby('cand_nm')
grouped.apply(get_top_amounts, 'contbr_occupation', n=7)
```

In [ ]:
```
grouped.apply(get_top_amounts, 'contbr_employer', n=10)
```

Bucketing Donation Amounts

In [ ]:
```
bins = np.array([0, 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000])
labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)
labels
```

In [ ]:
```
grouped = fec_mrbo.groupby(['cand_nm', labels])
grouped.size().unstack(0)
```

In [ ]:
```
bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)
bucket_sums
```

In [ ]:
```
normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)
normed_sums
```

In [ ]:
```
normed_sums[:-2].plot(kind='barh', stacked=True)
```

Donation Statistics by State

In [ ]:
```
grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])
totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)
totals = totals[totals.sum(1) > 100000]
totals[:10]
```

In [ ]:
```
percent = totals.div(totals.sum(1), axis=0)
percent[:10]
```